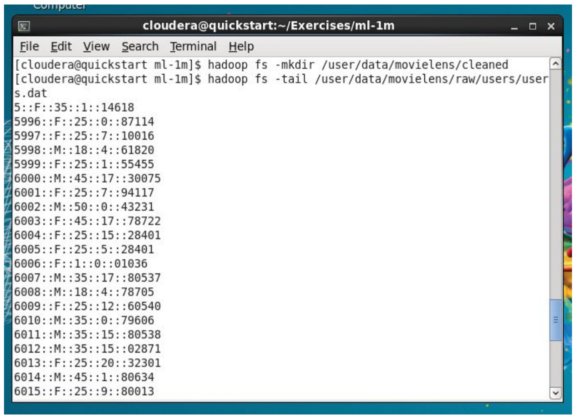
MSDS600 Week 8 Assignment - Nathan Worsham

Step 1: Introduction to Hadoop Commands and experiments with the Hadoop commands

Last week through trial and error I ended up learning about the hadoop file system in order to get my streaming example to work. There I learned that the HDFS is separate from the OS file system. So running through the file system commands this time felt more like a refresher.

I created the directories needed, put data in them, and confirmed the data. In the exercise it talks about tailing or cat -ing the file but in large files this is not good practice. I actually had an experience with this just recently where I had a folder that had grown accidentally to having 2.5 million files in it. Just doing a simple `ls` of the directory is not something the OS can really even handle well, in fact even deleting them individually presents problems. In this case I removed the parent directory as a work around, but this is the reason that large file systems like HDFS and GFS exist.


cloudera@quickstart:~/Exercises/ml-1m
File Edit View Search Terminal Help

```
[cloudera@quickstart ml-1m]$ hadoop fs -mkdir /user/data/movielens/cleaned
[cloudera@quickstart ml-1m]$ hadoop fs -tail /user/data/movielens/raw/users/user
s.dat
5::F::35::1::14618
5996::F::25::0::87114
5997::F::25::7::10016
5998::M::18::4::61820
5999::F::25::1::55455
6000::M::45::17::30075
6001::F::25::7::94117
6002::M::50::0::43231
6003::F::45::17::78722
6004::F::25::15::28401
6005::F::25::5::28401
6006::F::1::0::01036
6007::M::35::17::80537
6008::M::18::4::78705
6009::F::25::12::60540
6010::M::35::0::79606
6011::M::35::15::80538
6012::M::35::15::02871
6013::F::25::20::32301
6014::M::45::1::80634
6015::F::25::9::80013
```

Step 2: Preprocessing data using Python

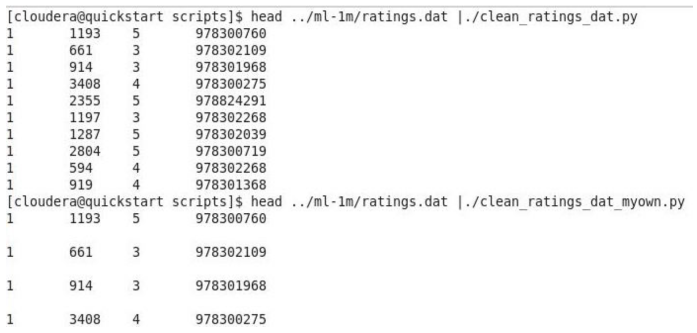
I started the creation of my python scripts with trying to recreate the `clean_ratings_dat.py`. The way I made my scripts differently is that I am used to coding Python in modules, so my scripts are made into functions and then the final main function brings everything together. This is considered the "pythonic way" and I find this is easier to troubleshoot large scripts because instead of having to comment out say 50 lines, you just comment out the one line that calls the function. When I was finished writing the script I tested it's output against the example scripts output by running the original (my working directory was the scripts directory):

```
head ../ml-1m/ratings.dat|./clean_ratings.dat
```

and then comparing that output with my own:

```
head ../ml-1m/ratings.dat|./clean_ratings_myown.dat
```

The problem I found here was that my script was giving an an extra blank line in-between each output whereas the original was not. I realized the original had a `line = line.strip()` which removes newline characters, so after adding this line of code my output then matched the example output.



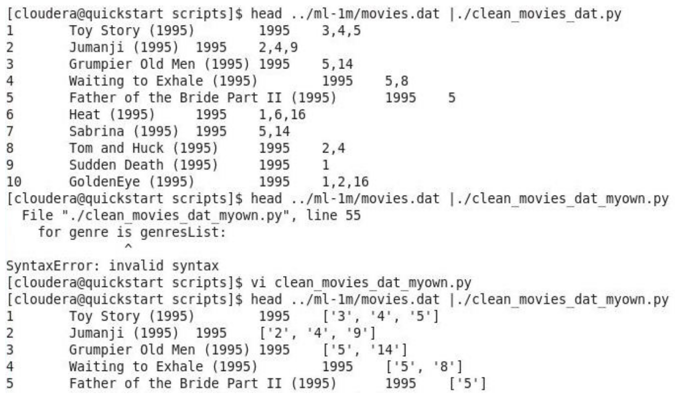
```
[cloudera@quickstart scripts]$ head ../ml-1m/ratings.dat |./clean_ratings_dat.py
1       1193    5       978300760
1       661     3       978302109
1       914     3       978301968
1       3408    4       978300275
1       2355    5       978824291
1       1197    3       978302268
1       1287    5       978302039
1       2804    5       978300719
1       594     4       978302268
1       919     4       978301368
[cloudera@quickstart scripts]$ head ../ml-1m/ratings.dat |./clean_ratings_dat_myown.py
1       1193    5       978300760

1       661     3       978302109

1       914     3       978301968

1       3408    4       978300275
```

For the `clean_movies_dat.py`, I only borrowed one section from the original which was the dictionary to convert the genres to the correct number. I then wrote my script made of three functions—one for converting genre to a number, one for pulling out the year using regex, and one to pull it all together. For my regex, I chose to be more specific than using `.*` as I have been bitten by that in the past—the star quantifier is considered "greedy"(regular-expressions.info, n.d.). I instead elected to just look for the open and close parenthesis and look specifically for four digits–`\(\d{4}\)`. Again running the example script first to compare output, I realized I had a syntax error, which after correcting that my output looked very similar except my list was printing out like a python list rather than a string.



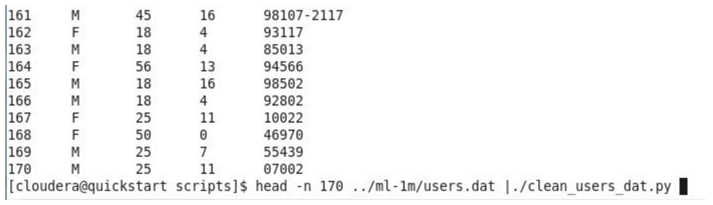
```
[cloudera@quickstart scripts]$ head ../ml-1m/movies.dat |./clean_movies_dat.py
1       Toy Story (1995)        1995    3,4,5
2       Jumanji (1995)  1995    2,4,9
3       Grumpier Old Men (1995) 1995    5,14
4       Waiting to Exhale (1995)        1995    5,8
5       Father of the Bride Part II (1995)      1995    5
6       Heat (1995)     1995    1,6,16
7       Sabrina (1995)  1995    5,14
8       Tom and Huck (1995)     1995    2,4
9       Sudden Death (1995)     1995    1
10      GoldenEye (1995)        1995    1,2,16
[cloudera@quickstart scripts]$ head ../ml-1m/movies.dat |./clean_movies_dat_myown.py
  File "./clean_movies_dat_myown.py", line 55
    for genre is genresList:
               ^
SyntaxError: invalid syntax
[cloudera@quickstart scripts]$ vi clean_movies_dat_myown.py
[cloudera@quickstart scripts]$ head ../ml-1m/movies.dat |./clean_movies_dat_myown.py
1       Toy Story (1995)        1995    ['3', '4', '5']
2       Jumanji (1995)  1995    ['2', '4', '9']
3       Grumpier Old Men (1995) 1995    ['5', '14']
4       Waiting to Exhale (1995)        1995    ['5', '8']
5       Father of the Bride Part II (1995)      1995    ['5']
```

So I added a line

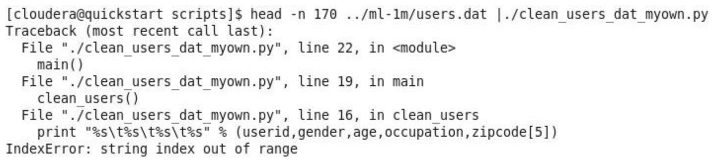
```
genresNumPrintList = ‘,’.join(genreNum)
```

and printed that instead. Now I had matching output.

For the final python script `clean_users_dat.py`, I found that that running a test by increasing the number of lines to display from the file until I could find an example zip code with the -XXXX format (called delivery sector and segment) appended and then running this through the example script, that the appended dash and 4 digits were still in the data even though the requirements of the assignment said to do otherwise. Running it through my code I had planned to only present the first 5 digits of the zip code to correct this issue but I found that I had forgotten to specify that I wanted a range—I got an error because I put `zipcode[5]` and indexes start with zero, so for many strings this character did not exist so I received the "index out of range" error. Correcting what I had meant to do I now found my code to work and successfully removed the extra zip code information.

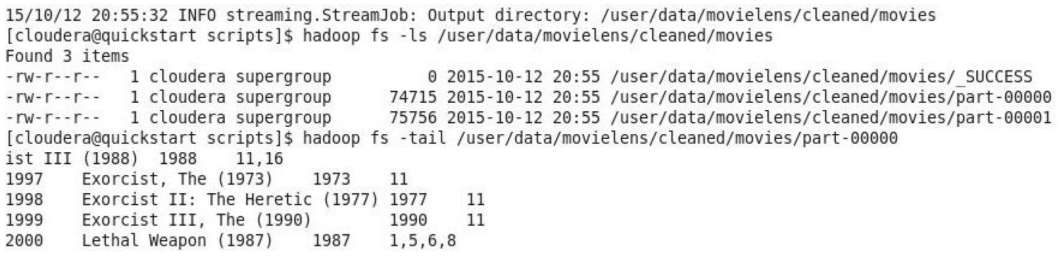


```
161     M       45      16      98107-2117
162     F       18      4       93117
163     M       18      4       85013
164     F       56      13      94566
165     M       18      16      98502
166     M       18      4       92802
167     F       25      11      10022
168     F       50      0       46970
169     M       25      7       55439
170     M       25      11      07002
[cloudera@quickstart scripts]$ head -n 170 ../ml-1m/users.dat |./clean_users_dat.py
```



```
[cloudera@quickstart scripts]$ head -n 170 ../ml-1m/users.dat |./clean_users_dat_myown.py
Traceback (most recent call last):
  File "./clean_users_dat_myown.py", line 22, in <module>
    main()
  File "./clean_users_dat_myown.py", line 19, in main
    clean_users()
  File "./clean_users_dat_myown.py", line 16, in clean_users
    print "%s\t%s\t%s\t%s" % (userid,gender,age,occupation,zipcode[5])
IndexError: string index out of range
```

For the bash script I really got my experience with the hadoop streaming command last week so interfacing with it was much better this go around. The one thing that threw me off at first was that there was no `-reduce` option like there was in last weeks example but the assignment says to use the option `-Dmapred.reduce.tasks=0` because the output of the mapper is the final output. Also the `STREAMING` variable needed to be updated to run the correct jar (which is why it is good that it is set as a variable since it is subject to change). I successfully ran a mapreduce of each option—ratings, users, and movies. After each run, I would confirm the contents of the output directory with `hadoop fs -ls directory` and then `hadoop fs -tail` a specific file in the directory to confirm its contents.



```
15/10/12 20:55:32 INFO streaming.StreamJob: Output directory: /user/data/movielens/cleaned/movies
[cloudera@quickstart scripts]$ hadoop fs -ls /user/data/movielens/cleaned/movies
Found 3 items
-rw-r--r--   1 cloudera supergroup          0 2015-10-12 20:55 /user/data/movielens/cleaned/movies/_SUCCESS
-rw-r--r--   1 cloudera supergroup      74715 2015-10-12 20:55 /user/data/movielens/cleaned/movies/part-00000
-rw-r--r--   1 cloudera supergroup      75756 2015-10-12 20:55 /user/data/movielens/cleaned/movies/part-00001
[cloudera@quickstart scripts]$ hadoop fs -tail /user/data/movielens/cleaned/movies/part-00000
ist III (1988)  1988    11,16
1997    Exorcist, The (1973)    1973    11
1998    Exorcist II: The Heretic (1977) 1977    11
1999    Exorcist III, The (1990)        1990    11
2000    Lethal Weapon (1987)    1987    1,5,6,8
```

Step 3: Create Hive tables

Following the HiveQL example, I queried `select count(*) from users where age<35;` to answer the how many users under 35, I received a response of 3421. To answer who the top three users who rated the most, I ended up using Stackoverflow.com's help. I was confused at first how to get this but by just looking at the data: `select * from ratings limit 25;` I could see that really these were lines that could be counted by userid because each time a user rated a different movie it is written to a new row. So using the query

```
select userid,count(*) as count from ratings group by userid order by count
DESC limit 3;
```

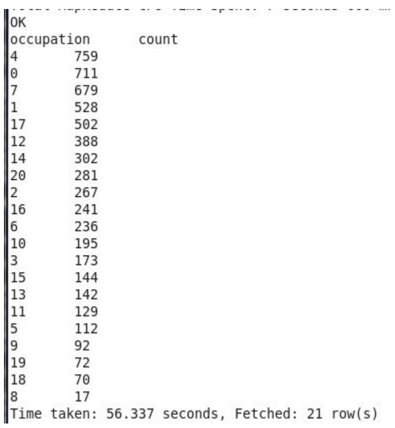
I received this answer for the top three users who rated the most:

```
4169    2314
1680    1850
4277    1743
```

With a total record of 1000209 ratings and 6040 users, these numbers seemed somewhat reasonable since 1000209/6040 is roughly 166, and you figure some users on rate once and some in the thousands. But I realized I could confirm this by querying a count of just my top user: `select count(*) from ratings where userid='4169';` and I received a match of 2314. After struggling with this question, the last question of how many users in each occupation I found to really be the same query as the second question because again I am just counting the number of rows by occupation. So using the query

```
select occupation,count(*) as count from users group by occupation order by
count DESC;
```

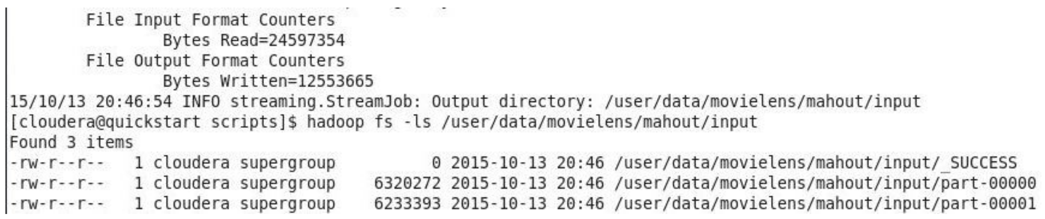
I received the answer of:



```
OK
occupation      count
4       759
0       711
7       679
1       528
17      502
12      388
14      302
20      281
2       267
16      241
6       236
10      195
3       173
15      144
13      142
11      129
5       112
9       92
19      72
18      70
8       17
Time taken: 56.337 seconds, Fetched: 21 row(s)
```

Step 4: Build the basic recommender system using Mahout

Running these scripts, I found again I needed to update the `STREAMING` jar variable. I ran the map reduce command, tried to create the directory (already existed) and uploaded the user file. Next I was able to successfully run the `do_mahout.sh` script.



```
        File Input Format Counters
                Bytes Read=24597354
        File Output Format Counters
                Bytes Written=12553665
15/10/13 20:46:54 INFO streaming.StreamJob: Output directory: /user/data/movielens/mahout/input
[cloudera@quickstart scripts]$ hadoop fs -ls /user/data/movielens/mahout/input
Found 3 items
-rw-r--r--   1 cloudera supergroup          0 2015-10-13 20:46 /user/data/movielens/mahout/input/_SUCCESS
-rw-r--r--   1 cloudera supergroup    6320272 2015-10-13 20:46 /user/data/movielens/mahout/input/part-00000
-rw-r--r--   1 cloudera supergroup    6233393 2015-10-13 20:46 /user/data/movielens/mahout/input/part-00001
```

Looking at the script, one thing I noticed was that it is set to use Pearson Correlation for its `-similarityClassname` setting. Looking at the output, I noticed that it only returns recommendations for the user when it is predicting the user would rate the movie a 5. Taking user 1 as an example,

```
select * from ratings join movies on (ratings.movieid = movies.id)
where ratings.userid='1' and rating='5';
```

we see that some of the movies this user rated a 5 were "One Flew Over the Cuckoo's Nest", "Ben-Hur", "A Bug's Life", and "Dog Park". Likewise grabbing a handful of the films that have been recommended,

```
select * from movies where id='1146' or id='1012' or id='1937' or '2133';
```

The sample titles recommended: "Old Yeller", "Curtis's Charm", "Going My Way", and "Adventures in Babysitting". Now of course the user rated a lot more than I just gave as an example but are these good recommendations? I suppose you would have to ask user 1. Though from a critical acclaim point-of-view, the 4 movies I chose which user 1 rated are all highly rated on rottentomates.com (88-95%) with the exception of "Dog Park", which only scores a 36%. So user 1 (and probably everyone) clearly has some eclectic tastes.

References

Decalage.info, n.d. Retrieved from http://www.decalage.info/en/python/print_list
Soniak, Matt. 2013. Retrieved from http://mentalfloss.com/article/53384/what%E2%80%99s-deal-those-last-4-digits-zip-codes
Stackoverflow.com, n.d. Retrieved from http://stackoverflow.com/questions/9994970/counting-in-hadoop-hive
Regular-expressions.info, n.d. Retrieved from http://www.regular-expressions.info/repeat.html
Rottentomatoes.com, n.d. Retrieved from http://www.rottentomatoes.com/m/dog_park